FIG. 1

```
class ClassA{                                          [1]
public:
   void Function1(void){…};
   void Function2(void){…};                            [6]
};

class ClassB : public ClassA{                          [3]
public:
   void Function2(void){…};                            [7]
   void Function3(void){…};
};

class ClassC{
public:
   virtual void Function1(void)=0;                     [8]
};

class ClassD: public ClassC{
Public:
   void Function1(void);                               [10]
};

main(void){
   ClassA ObjectA;                                     [2]
   ObjectA.Function1();
   ObjectA.Function2();

   ClassB ObjectB;                                     [4]
   ObjectB.Function1();
   ObjectB.Function2();
   ObjectB.Function3();

   ClassA *PointerA;                                   [5]
   PointerA=&ObjectB;
   PointerA->Function1();
   PointerA->Function2();
 //PointerA->Function3(); //Because ClassA does not have Function3,
                          //the compliler outputs an error.
 //ClassCObjectC;         //Because ClassC has a pure virtual function,
                          //an object cannot be produced.
                    [9]   //Therefore, the compliler outputs an error.
 ClassD ObjectD;
 }
```

FIG. 2

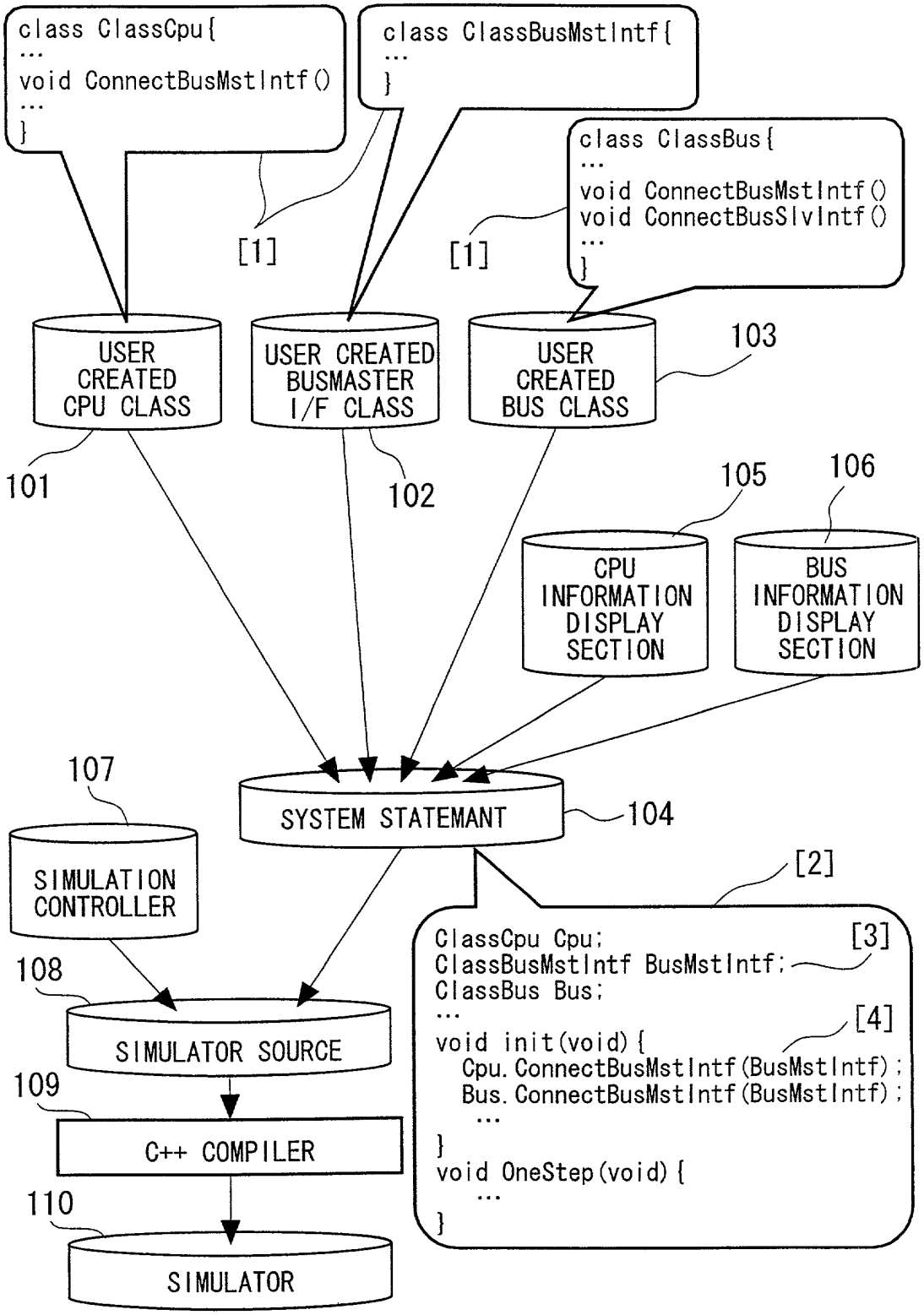
class ClassCpu{
...
void ConnectBusMstIntf()
...
}
class ClassBusMstIntf{
...
}
class ClassBus{
...
void ConnectBusMstIntf()
void ConnectBusSlvIntf()
...
}
[1]
[1]
USER CREATED CPU CLASS
USER CREATED BUSMASTER I/F CLASS
USER CREATED BUS CLASS
103
101
102
105
106
CPU INFORMATION DISPLAY SECTION
BUS INFORMATION DISPLAY SECTION
107
SYSTEM STATEMANT
104
SIMULATION CONTROLLER
[2]
ClassCpu Cpu;
ClassBusMstIntf BusMstIntf;
[3]
ClassBus Bus;
...
[4]
void init(void){
Cpu.ConnectBusMstIntf(BusMstIntf);
Bus.ConnectBusMstIntf(BusMstIntf);
...
}
void OneStep(void){
...
}
108
SIMULATOR SOURCE
109
C++ COMPILER
110
SIMULATOR

FIG. 3A

EXAMPLE OF DESCRIPTION IN C++

```
class ClassA{
    ...
    void GetData(int i);
    ...
};

class ClassB{
    ...
    ClassA *PointerA;
    void ConnectA(ClassA *a)
        {PointerA=a};
    void OneStep(void){
        ...
        ...=PointerA->GetData(i);
        ...
    }
};

main(){
    ClassA ObjectA;
    ClassB ObjectB;
    ObjectB.ConnectA(&ObjectA);
    ...
    ObjectB.OneStep();
    ...
}
```

FIG. 3B

SCHEMATIC DIAGRAM

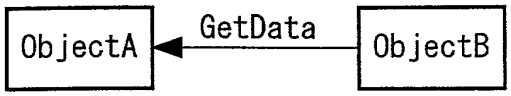

FIG. 4

CLASS	PROPERTY
CmComponent	general circuit
CmSyncModule	circuit operating synchronously with clock
CmBusMaster	main section of bus master
CmBusSlave	main section of bus slave
CmBusMstIntf	bus master interface
CmBusSlvIntf	bus slave interface
CmBusSystem	bus
CmCpu	CPU
CmMemory	memory
CmHier	hierarchy of a circuit containing the bus

FIG. 5

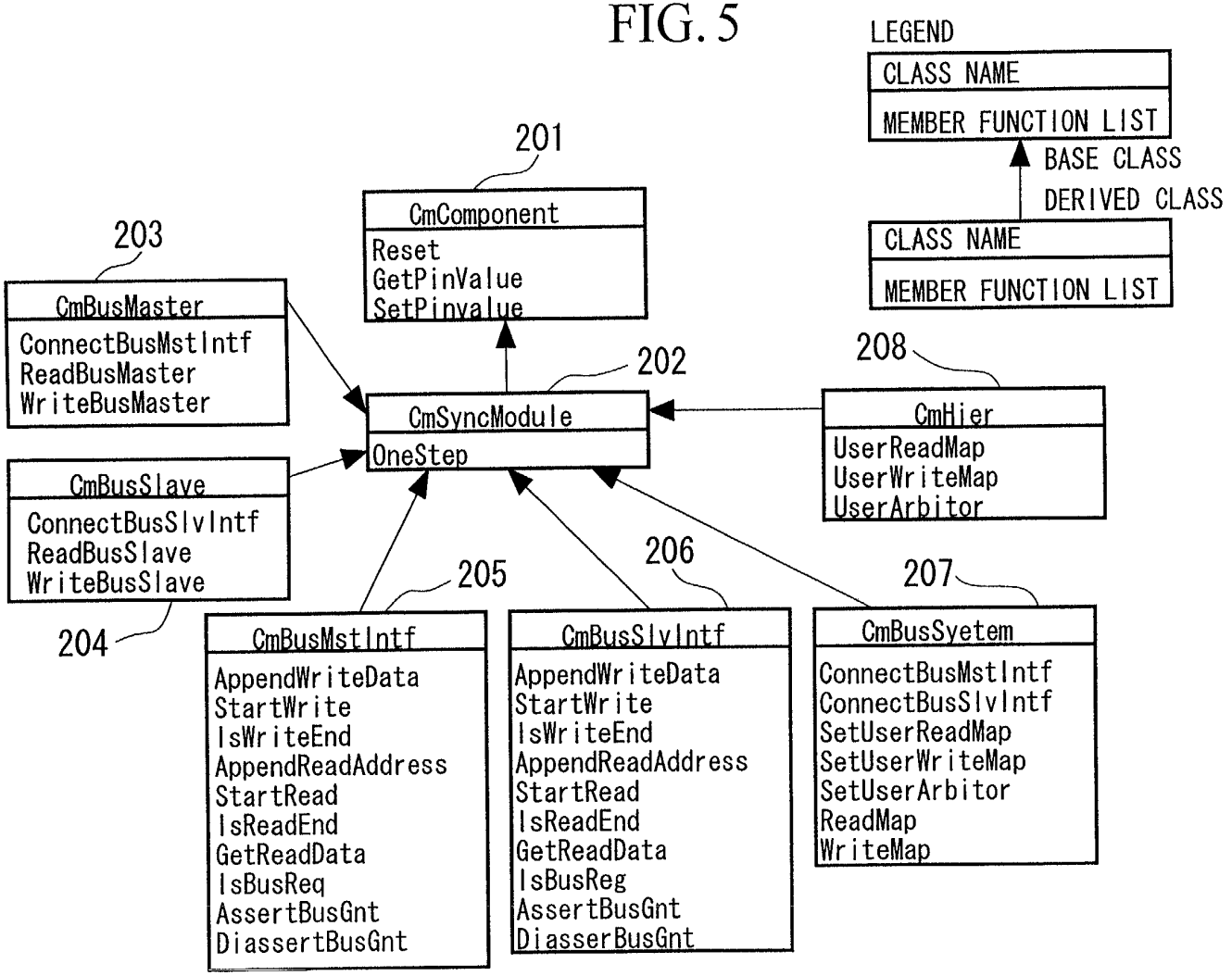
LEGEND
CLASS NAME
MEMBER FUNCTION LIST
BASE CLASS
DERIVED CLASS
CLASS NAME
MEMBER FUNCTION LIST
201
CmComponent
Reset
GetPinValue
SetPinvalue
203
CmBusMaster
ConnectBusMstIntf
ReadBusMaster
WriteBusMaster
202
CmSyncModule
OneStep
208
CmHier
UserReadMap
UserWriteMap
UserArbitor
CmBusSlave
ConnectBusSlvIntf
ReadBusSlave
WriteBusSlave
204
205
206
207
CmBusMstIntf
AppendWriteData
StartWrite
IsWriteEnd
AppendReadAddress
StartRead
IsReadEnd
GetReadData
IsBusReq
AssertBusGnt
DiassertBusGnt
CmBusSlvIntf
AppendWriteData
StartWrite
IsWriteEnd
AppendReadAddress
StartRead
IsReadEnd
GetReadData
IsBusReg
AssertBusGnt
DiasserBusGnt
CmBusSyetem
ConnectBusMstIntf
ConnectBusSlvIntf
SetUserReadMap
SetUserWriteMap
SetUserArbitor
ReadMap
WriteMap

FIG. 6

ACTUAL CIRCUIT CLASS	CIRCUIT BASE CLASS
CmPciBusMstIntf	CmBusMstIntf
CmPciBusSlvIntf	CmBusSlvIntf
CmPciBusSystem	CmBusSystem
CmV850	CmCpu

FIG. 7

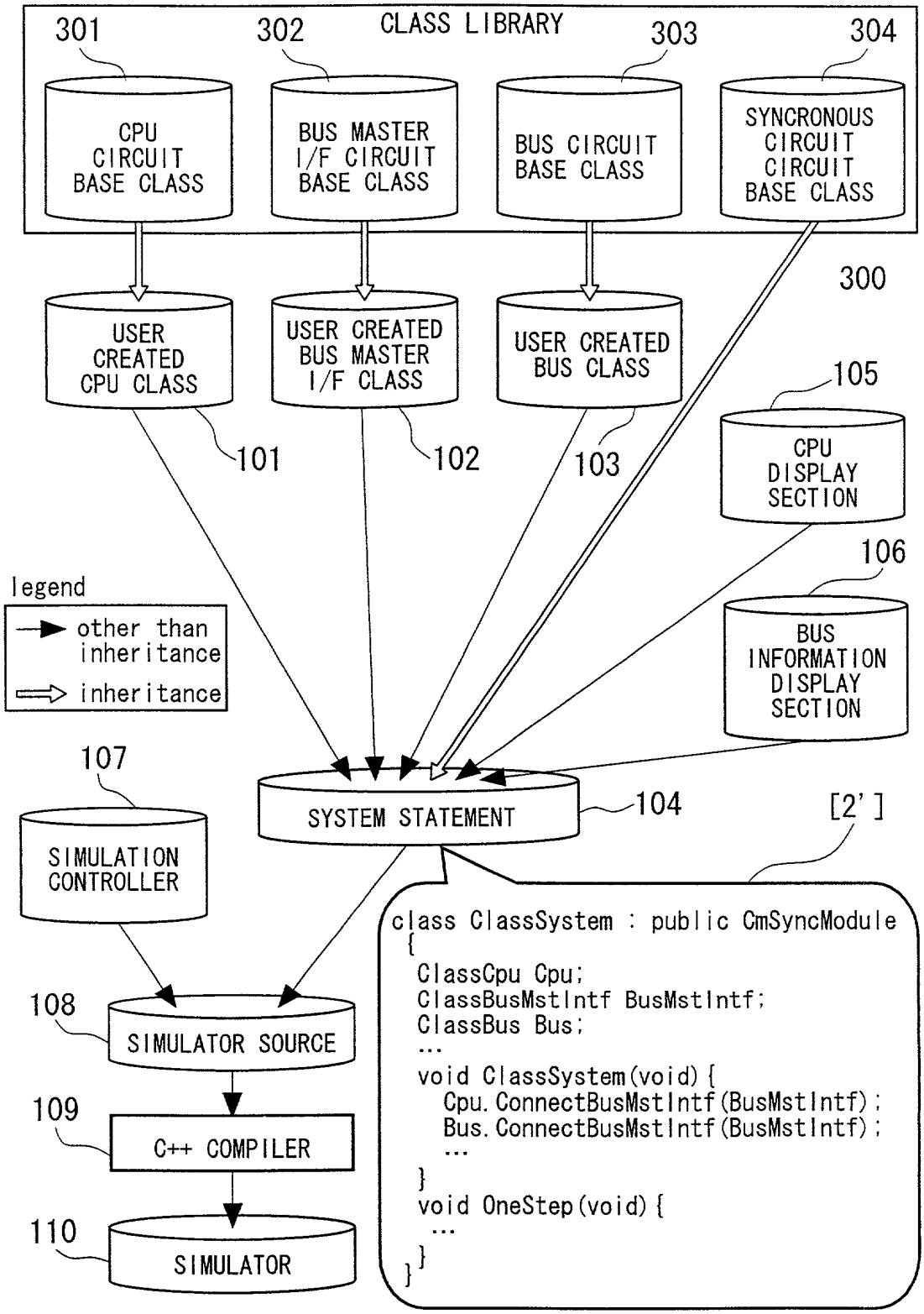
CLASS LIBRARY
301
302
303
304
CPU CIRCUIT BASE CLASS
BUS MASTER I/F CIRCUIT BASE CLASS
BUS CIRCUIT BASE CLASS
SYNCRONOUS CIRCUIT CIRCUIT BASE CLASS
300
USER CREATED CPU CLASS
USER CREATED BUS MASTER I/F CLASS
USER CREATED BUS CLASS
101
102
103
105
CPU DISPLAY SECTION
106
BUS INFORMATION DISPLAY SECTION
legend
other than inheritance
inheritance
107
SYSTEM STATEMENT
104
[2']
SIMULATION CONTROLLER
class ClassSystem : public CmSyncModule
{
ClassCpu Cpu;
ClassBusMstIntf BusMstIntf;
ClassBus Bus;
...
void ClassSystem(void){
Cpu.ConnectBusMstIntf(BusMstIntf);
Bus.ConnectBusMstIntf(BusMstIntf);
...
}
void OneStep(void){
...
}
}
108
SIMULATOR SOURCE
109
C++ COMPILER
110
SIMULATOR

FIG. 8

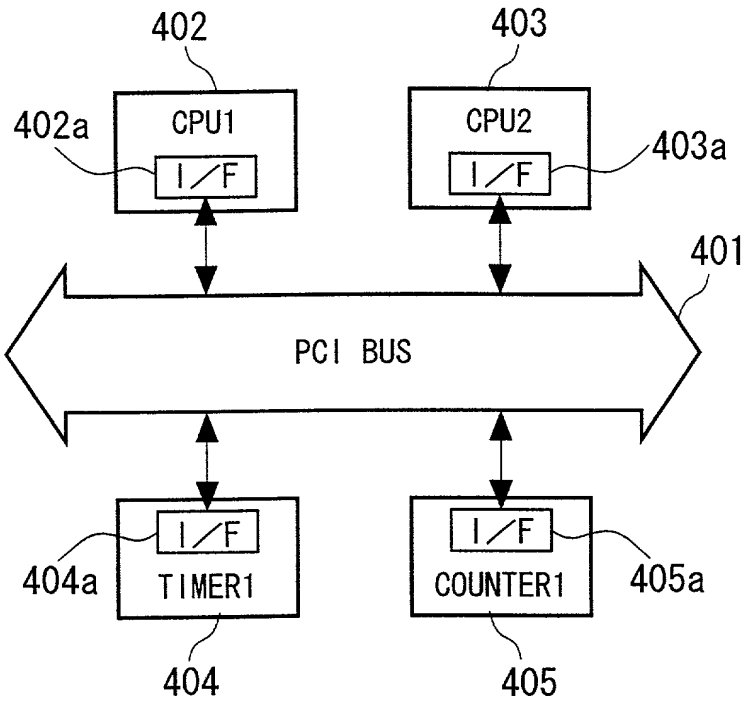
402
403
CPU1
CPU2
I/F
I/F
402a
403a
401
PCI BUS
I/F
I/F
TIMER1
COUNTER1
404a
405a
404
405

FIG. 9

```
class ABC : public CmHier {                                    [1]

 // Create a PCI bus in a system ABC.                          [2]
 CmPciBusSystem PciBusSystem;

 // Create two CPU, which are Cpu1, and Cpu2, in the system ABC  [3]
 CmCpu Cpu1, Cpu2;

 // Because Cpu1, and Cpu2 are bus masters,                     [4]
 // a bus master interface for connecting Cpu1 and Cpu2 to the bus
 // are created in the system ABC.
 CmPciBusMstIntf Cpu1BusMstIntf, Cpu2BusMstIntf;

 // Create a circuit Timer1 in the system ABC.                  [5]
 CmTimer Timer1;

 // Because Timer1 is a bus slave,                              [6]
 // a bus slave interface for connecting Timer1 to the bus
 // is created in the system ABC.
 CmPciBusSlvIntf Timer1BusSlvIntf;

 // Create a circuit Counter1 in the system ABC.                [7]
 CmCounter  Counter1;

 // Because Counter1 is a bus slave,                            [8]
 // a bus slave interface for connecting Counter1 to the bus
 // is created in the system ABC.
 CmPciBusSlvIntf Counter1BusSlvIntf;

 ...  // Other modules are described (omitted).                 [9]
```

FIG. 10

```
// UserBusReadMap describes a bus read map.
// The return value is structure RDATA                    [1]
// RDATA.Status returns 0 on success and 1 on failure.
// RDATA.Data returns the read value on success.
// ULONG is a signal value of 32 bits.
RDATA UserBusReadMap(ULONG address, int byte_count) {
// Based on address, the circuit from which the value
// is to read is determined.
// Execute the read function of the bus slave interface of the circuit
    RDATA v;                                               [2]

    // Read values from addresses 100 to 200 into Timer1
    if ((100 <= address) && (address < 200)) {             [3]

     // Designate the address to be read                   [4]
      Timer1BusSlvIntf.AppendReadAddress(0, address-100,
                                                byte_count);

     // Read the value.                                    [5]
        Timer1BusSlvIntf.StartRead(0);

     // Check whether the reading is successful            [6]
        if (Timer1.IsReadEnd(0)) {

  // The reading is successful.                            [7]
   v.Status = 0;

   // Extract the read value.                              [8]
    v.Data = Timer1BusSlvIntf.GetReadData(0);
    return v;
        } else {

  // The reading failed.                                   [9]
  v.Status = 1;
  return v;
      }
   }                                                       [10]
  // Read the values from addresses 200 to 300 in Counter1.
  else if ((200 <= address) && (address < 300)) {
  // The similar steps as those for
  // Timer1 are repeated in the following.                 [11]
  } else if ... // The same steps are repeated for the other circuits
}                                                         [12]
```

FIG. 11

```
// UserBusWriteMap describes a bus write map.                    [1]
// The return value is int.
// Returns 0 on success, and 1 on failure.
// ULONG is a signal value of 32 bits.
int UserBusWriteMap(ULONG address, ULONG data,
      int byte_count) {
                                                                  [2]
// Write a value, based on address.
// Determine a circuit, and execute the write function
// of the bus slave interface of the determined circuit.          [3]

  // Write the values into addresses 100 to 200 of Timer1
    if ((100 <= address) && (address < 200)) {                    [4]

       // Designate the address into which the value to be written.
        Timer1BusSlvIntf.AppendWriteData(0, address-100,
                                         data, byte_count);
                                                                  [5]
        // Write the value.
        Timer1BusSlvIntf.StartWrite(0);
                                                                  [6]
        // Check whether the writing is successful.
        if (Timer1.IsWriteEnd(0)) {

  // The writing is successful.                                   [7]
  return 0;
      } else {

  //The writing failed.                                          [8]
  return 1;
      }
    }
                                                                  [9]
// Write the values into the addresses 250 to 350 from Counter1.
//The addresses are different for the reading and the writing.
    else if ((250 <= address) && (address < 350)) {
// In the following, similar steps to those for                  [10]
// Timer1 are repeated.

} else if ... // Similar steps are repeated                      [11]
              // for the other circuits.
}
```

FIG. 12

```
// UserArbitor describes an arbiter.                    [1]
// The return value is void (no return value).
// The permission to use the bus is directly given to
// the bus master interface.
void UserArbitor(void) {

    // The request to use the bus is directly extracted  [2]
    // from the bus master interface.
    int Cpu1BusReq = Cpu1BusMstIntf.IsBusReq();
    int Cpu2BusReq = Cpu2BusMstIntf.IsBusReq();

    // Cpu1 is preferred to Cpu2.                        [3]
    if (Cpu1BusReq) {

// Allow Cpu1 to use the bus when Cpu1 requests to use the bus   [4]
// even if the Cpu2 requests to use the bus
        Cpu1BusMstIntf.AssertBusGnt();
        Cpu2BusMstIntf.DiassertBusGnt();

    } else if (Cpu2BusReq) {

    // Allow Cpu1 to use the bus when Cpu1 requests to use the bus   [5]
    // even if the Cpu2 requests to use the bus

        Cpu1BusMstIntf.DiassertBusGnt();                  [6]
        Cpu2BusMstIntf.AssertBusGnt();
    } else {
        Cpu1BusMstIntf.DiassertBusGnt();                  [7]
        Cpu2BusMstIntf.DiassertBusGnt();
    }
}
```

FIG. 13

```
// ABC(void) is a function (constructor)
// when creating the object of the system ABC.          [1]
ABC(void) {

    // Connect the bus with the circuits.               [2]
    Cpu1.ConnectBusMstIntf(&Cpu1BusMstIntf);
    Cpu2.ConnectBusMstIntf(&Cpu2BusMstIntf);
    Timer1.ConnectBusSlvIntf(&Timer1BusSlvIntf);
    Counter1.ConnectBusSlvIntf(&Counter1BusSlvIntf);
    PciBusSystem.ConnectBusMstIntf(&Cpu1BusMstIntf);
    PciBusSystem.ConnectBusMstIntf(&Cpu2BusMstIntf);
    PciBusSystem.ConnectBusSlvIntf(&Timer1BusSlvIntf);
    PciBusSystem.ConnectBusSlvIntf(&Counter1BusSlvIntf);
    ... // Describe the other circuits (omitted).      [3]

    // Designate an address map.  Register the created address map   [4]
    // in PciBusSystem.  Thus, PciBusSystem can use
    // the created address map.
    PciBusSystem.SetUserReadMap(UserReadMap);
    PciBusSystem.SetUserWriteMap(UserWriteMap);
}
// OneStep describes the operation of the system ABC                  [5]
// within one unit time.
 void OneStep(void) {                                               [6]
    // The system ABC is operated by the arbitrary behaviors
    //  of the components. Accordingly, in the level of the
    // system ABC, the respective components in the system
    // are simply operated in one unit time.
     PciBusSystem.OneStep();                                        [7]
    // Although the bus does not require
    // the clock because the bus is a signal line, the clock
    // is given to the bus in order to extract data synchronously
    // with the clock for debugging.
    Cpu1BusMstIntf.OneStep();
    Cpu2BusM stIntf.OneStep();                                      [8]
    Timer1BusSlvIntf.OneStep();
    Counter1BusSlvIntf.OneStep();
    Cpu1.OneStep();
    Cpu2.OneStep();
    Timer1.OneStep();
    Counter1.OneStep();
    ... // The other circuits are operated.                       [9]
  }
}
```

FIG. 14

```
class CmComponent {
  // Usage: describe a sequence circuit with a clock.


public:
    virtual void Reset(void) = 0;
      // Usage: asynchronous resetting.

virtual RDATA GetPinValue(int no) {
      // Usage: return the signal value at the terminal number "no".
    }

virtual int SetPinValue(int no, ULONG data);
      // Usage: apply data at the terminal number "no".
   }
 };
```

FIG. 15

```
class CmSyncModule : public CmComponent {
// Usage: describe a sequence circuit having a clock.
// Note: CmSyncModule uses Reset, GetPinValue, and SetPinValue
//       defined in CmComponent as are.

public:
   virtual void OneStep(void) = 0;
// Usage: operate the circuit by one step.
};
```

FIG. 16

```
class CmBusMaster : public CmSyncModule {
  // Usage: describe the main section of the bus master.

public:
  virtual void ConnectBusMstIntf(CmBusMstIntf *bus_mst_intf) {
    // Usage: connect the circuit to the bus master interface.
  }

  virtual RDATA ReadBusMaster(ULONG address,
     int byte_count) = 0;
    // Usage: read a value from the circuit to the bus.

  virtual int WriteBusMaster(ULONG address, ULONG data,
     int byte_count) = 0;
    // Usage: write a value from the object into the bus.
};
```

FIG. 17

```
class CmBusSlave : public CmSyncModule {
 // Usage: describe the main section of the bus slave.

public:
  virtual void ConnectBusSlvIntf(CmBusMstIntf *bus_mst_intf) {
   // Usage: connect the circuit to the bus slave interface.
  }

  virtual RDATA ReadBusSlave(ULONG address, int byte_count) = 0;
   // Usage: read the value from the circuit to the bus.

  virtual int WriteBusSlave(ULONG address, ULONG data,
                                          int byte_count) = 0;
   // Usage: write the value from the bus into the object.
};
```

FIG. 18

```
class CmBusMstIntf : public CmSyncModule {
// Usage: create a bus master interface.

public:
  virtual bool IsBusReq(void) {
// Usage: returns a value indicating whether the bus master
//        interface requests to use the bus.
  }
  virtual void AssertBusGnt(void) {
// Usage: notify the bus master interface of the permission
//        use the bus, which is sent from an external circuit
//        (normally, the arbiter).
  }
  virtual void DiassertBusGnt(void) {
// Usage: notify the bus master interface of the rejection
//       to use the bus, which is sent from an external circuit
//       (normally, the arbiter).
  }
  virtual void AppendWriteData(int mode, ULONG address,
                               ULONG data, int byte_count) {

// Usage: when mode is 0, register address which is to be written
//     into the circuit module through the bus, data, and the
//     number of bytes byte_count, in the buffer of the circuit
  }
  virtual void StartWrite(int mode) {
// Usage: when mode is 0, start writing data into the bus.
  }
  virtual bool IsWriteEnd(int mode) {
// Usage: when mode is 0, return a value
//       indicating whether the writing is completed.
  }
  virtual void AppendReadAddress(int mode,
                                 ULONG address,  int byte_count) {
// Usage: when mode is 0, register address which is to be
//        written into the circuit module through the bus,
//        data, and the number of bytes byte_count,
//        in the buffer of the circuit

  }
```

FIG. 19

```
(continued from the definition of class CmBusMstIntf)


  virtual void StartRead(int mode) {
// Usage: when mode is 0, start reading the data from the bus.
  }
  virtual bool IsReadEnd(int mode) {
// Usage:  when mode is 0, return a value indicating whether
//       the reading operation from the bus is completed.
  }
  virtual ULONG GetReadData(int mode){
// Usage: when mode is 0, and when the reading from the bus is
//      completed, extract data which has been read and
//      stored in the buffer and return the data to the bus master.
  }
  virtual void OneStep(void) {
// Usage: operate the circuit by one step.
// Operation: while in the reading or writing operation from the
//         bus master to the bus, extract one data, to be read
//          or written, from the buffer, and send it to the bus.
//         When the buffer becomes empty,
//         the reading or writing is completed.
// Supplement: OneStep function defined in
//                  CmSyncModule is overridden.
  }
};
```

FIG. 20

```
class CmBusSlvIntf : public CmSyncModule {
public:
  virtual bool IsBusReq(void) {
  // Usage: return a value indicating whether the bus
  //        slave interface requests to use the bus.
   }
  virtual void AssertBusGnt(void) {
// Usage: the external circuit (normally, the arbiter) notifies the
//        bus slave interface of the permission to use the bus.
  }
  virtual void DiassertBusGnt(void) {
// Usage: the external circuit (normally, the arbiter)
//        notifies the bus slave interface of the prohibition
//        to use the bus.
   }
  virtual void AppendWriteData(int mode, ULONG address,
                               ULONG data, int byte_count) {
// Usage: when mode is 0, register address which is to be
//       written into the bus slave, data, and the number of
//       bytes byte_count, into the buffer of the circuit
  }
  virtual void StartWrite(int mode) {
// Usage: when mode is 0, write the data into the bus slave,
//       based on the data registered in the buffer.
  }
  virtual bool IsWriteEnd(int mode) {
// Usage: when mode is 0, return a value indicating whether the
//        writing operation into the bus slave completed.
   }
  virtual void AppendReadAddress(int mode, ULONG address,
                                 int byte_count) {
// Usage: when mode is 0, the address, which is to be read from
//        the bus slave, and the number of bytes byte_count into
//        the buffer in the circuit.
  }
```

FIG. 21

```
  (continued from the definition of class CmBusSlvIntf)

  virtual void StartRead(int mode) {

   // Usage: when mode is 0, read the data from the bus slave,
   }     // based on the data registered in the buffer.
  virtual bool IsReadEnd(int mode) {
// Usage: when mode is 0, return a value indicating whether
      // the reading operation from the bus slave is completed.
  }
  virtual ULONG GetReadData(int mode){
Usage: when mode is 0, and when the reading operation from the
//     bus slave is completed, extract data which has been read
//     from the bus slave and has been stored in the buffer,
//     and return it to the bus.
  }
  virtual void OneStep(void) {

// Usage: operate the circuit by one step.
// Operation: OneStep does not perform operation
//           in connection with the bus slave.
// Supplement: CmBusSlvInt overrides OneStep defined in SyncModule.

  }
};
```

FIG. 22

```
class CmBusSystem : public CmSyncModule {
public:
  virtual void ConnectBusMstIntf(CmBusMstIntf *bus_mst_intf) {
  // Supplement: CmBusSlvInt overrides OneStep defined in SyncModule.
  }
  virtual void ConnectBusSlvIntf(CmBusSlvIntf *bus_slv_intf) {
  // Usage: connect the bus slave interface to the bus.
  }
  virtual void SetUserReadMap(CmHier *hier) {
  // Usage: designate a bus read map.  When sending the object of
  //      CmHier class, the UserReadMap function of CmHier object is
  //      registered as the bus read map of the bus.

  }
  virtual void SetUserWriteMap(CmHier *hier) {
  // Usage: designate a bus write map.  When sending the object of
  //       CmHier class, the UserWriteMap function of CmHier object is
  //       registered as the bus write map of the bus.
  }
  virtual void SetUserArbitor(CmHier *hier) {
  // Usage: designate an arbiter.  When sending the object of
  //       CmHier class, the UserArbitor function of CmHier object is
  //       registered as the arbiter of the bus.
  }
  virtual RDATA ReadMap(ULONG address, int byte_count) {
  // Usage: read data from a circuit module through the bus.
  //     Based on the given address, the circuit which is the target to be
  //     read is determined.  The data of the number of bytes byte_count are
  //     read from the circuit module, and the read data is returned.
  }
  virtual int WriteMap(ULONG address, ULONG data, int byte_count) {
  // Usage: write data into a circuit module through the bus.
  //     Based on the given address, the circuit to which the data is to be
  //     written is determined.  The data of the number of bytes byte_count
  //     are written into the circuit module, and a value indicating whether
  //     the writing is successful is returned.

   }
  virtual void OneStep(void) {
  // Usage: operate the bus arbiter by one step.
  //      Operate the information display function of the bus.
  // Supplement: because the bus is a simple signal line,
  //       the bus does not operate in response to the clock.
  }
};
```

FIG. 23

```
class CmHier : public CmSyncModule {
  virtual RDATA UserReadMap(ULONG address, int byte_count) {
   // Usage: describe the bus read map of the bus
   //          included in the system.
  }
  virtual int UserWriteMap(ULONG address, ULONG data,
                                                  int byte_count) {
   // Usage: describe the bus write map of the bus
   //      included in the system.
  }
  virtual void UserArbitor(void) {
   // Usage: describe the arbiter of the bus
   //      included in the system.
  }
}
```

FIG. 24

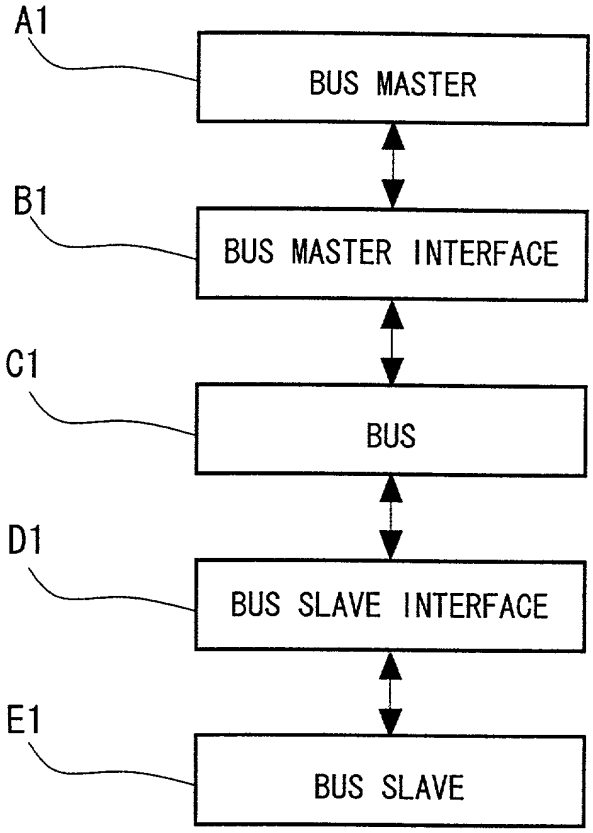

FIG. 25

ADDRESS	DATA	NUMBER OF BYTES
0x00100000	0x00112233	4
0x00100004	0x44556677	4

FIG. 26

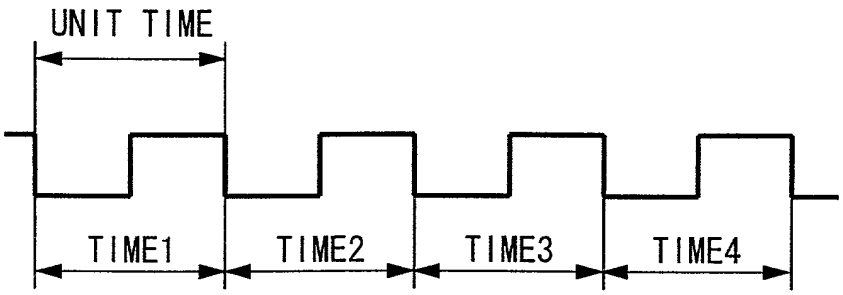

FIG. 27

```
# time1
    OneStep of A1
      A1 executes AppendWriteData(0, 0x00100000, 0x00112233, 4)of B1.
        B1 stores the data into the buffer.
      A1 executes AppendWriteData(0, 0x00100004, 0x44556677, 4)of B1.
        B1 stores the data into the buffer.
      A1 executes StartWrite(0) of B1.
        B1 memorizes the write request.
      A1 executes IsWriteEnd(0) of B1.
        B1 returns a value indicating that B1 is writing the data.
      A1 goes on standby.
    OneStep of B1
      Because B1 is requested by A1 to write the data,
      B1 requests to use the bus.
      When the IsBusReq function of the bus master interface B1 is
      externally executed, the return value is true.
    OneStep of C1
      Operate the arbiter.
      Arbiter executes IsBusReq() of B1.
       B1 returns that B1 is requesting to use the bus.
      Arbiter permits B1 to use the bus.
      Arbiter executes AssertBusGnt() of B1.
        B1 memorizes the permission to use the bus.
    OneStep of D1
     no operation
    OneStep of E1
    The special operation of the bus slave E1 is performed,
    but does not includes an operation in connection with the bus.
time
```

FIG. 28

```
#time2
    OneStep of A1
      A1 executes IsWriteEnd(0) of B1.
        B1 returns that B1 is writing the data.
      A1 goes on standby.
    OneStep of B1
      Because B1 is permitted to use the bus,
              B1 writes the data into the bus.
      That is, the following operations are performed.
        B1 extracts one data from the buffer.
          address:0x00100000  data:0x00112233  number of bytes:4
        B1 executes WriteMap(0x00100000,0x00112233,4) of C1.
          WriteMap function of C1 executes the following operations.
            Execute AppendWriteData(0,0x00100000,0x00112233,4)
            function of D1.
              D1 stores the given data into the buffer.
            D1 executes StartWrite(0) function.
              D1 executes WriteBusSlave(0x00100000,0x00112233,4)
              function of E1.
                E1 internally writes the given data,
                  and returns a value,indicating whether the
                  writing is successful, to D1.
                 In the example, the writing is successful.
            Execute IsWriteEnd(0) function of D1.
              Because the writing operation by D1 into E1 is
              successful, the value is true.
            Return the return value of IsWriteEnd(0) function D1.
        B1 comes to know that the data has been successfully written,
        based on the return value.
        Then, B1 deletes data from the buffer.
        Because B1 stores data, the same process is performed
        in the next OneStep of B1.
    OneStep of C1
      no operation
    OneStep of D1
      no operation
    OneStep of E1
      The special operation of the bus slave E1 is performed,
      but does not includes an operation in connection with the bus.
time
```

FIG. 29

```
#time3
    OneStep of A1
      A1 executes IsWriteEnd(0) of B1.
        B1 returns that B1 is writing the data.
      A1 stays on standby.
    OneStep of B1
      Because B1 is permitted to use the bus, B1 writes the
      data into the bus.
      That is, the following operations are performed.
        B1 extracts one data from the buffer.
          address: 0x00100004  data: 0x44556677 number of bytes: 4
        B1 executes WriteMap(0x00100004,0x44556677,4) of C1.
          WriteMap function of C1 performs the following operations.
            Execute AppendWriteData(0,0x00100004,0x44556677,4)
            function of D1.
              D1 stores the given data into the buffer.
            Execute StartWrite(0) function of D1.
              D1 executes WriteBusSlave(0x00100004,0x44556677,4)
              function of E1.
                E1 internally writes the given data, and returns a value,
                indicating whether the writing is successful, to D1.
                In the example, the writing is successful.
            Execute IsWriteEnd(0) function of D1.
              Because the writing from D1 into E1 has been successful,
              the return value is true.
            Return IsWriteEnd(0) function of D1.
        B1 comes to know that the data has been successfully written,
        based on the return value.
        Then, B1 deletes one data from the buffer.
        There is no data in the buffer of B1.
        B1 completes the writing operation.
        The return value of IsWriteEnd(0) function of B1 is true.
    OneStep of C1
      no operation
    OneStep of D1
      no operation
    OneStep of E1
      The special operation of the bus slave E1 is performed,
      but does not includes an operation in connection with the bus.
time
```

FIG. 30

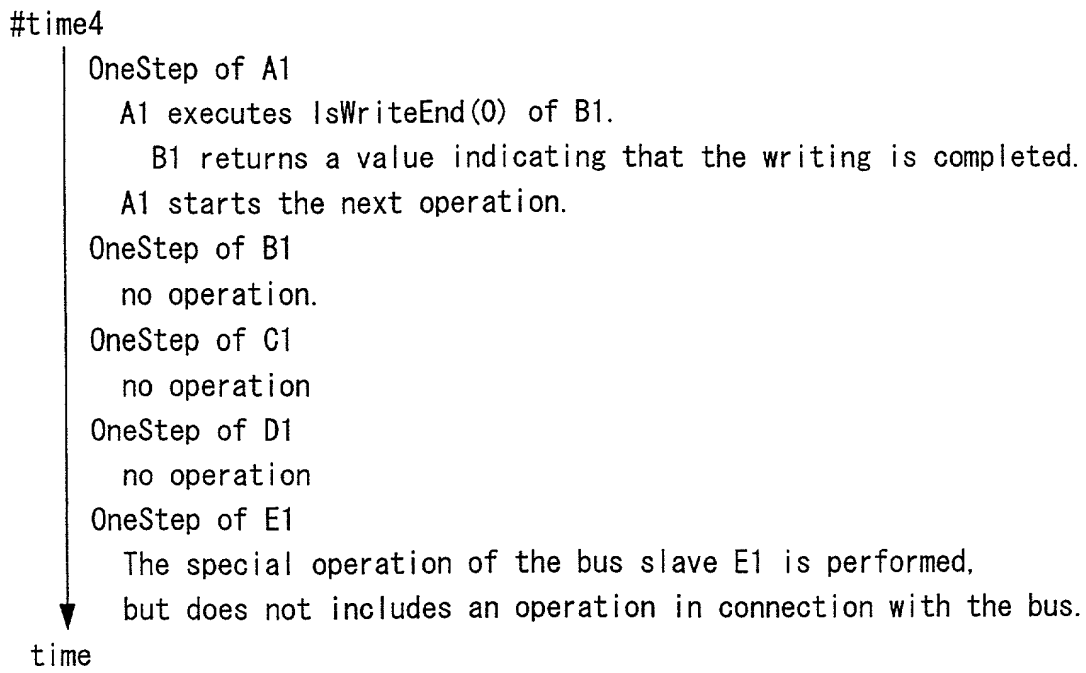
#time4
OneStep of A1
A1 executes IsWriteEnd(0) of B1.
B1 returns a value indicating that the writing is completed.
A1 starts the next operation.
OneStep of B1
no operation.
OneStep of C1
no operation
OneStep of D1
no operation
OneStep of E1
The special operation of the bus slave E1 is performed,
but does not includes an operation in connection with the bus.
time

FIG. 31A

Statement of ClassX

```
class ClassX{                                                    [1]
   CmBusMstIntf *BusMstIntf;
   void ConnectBusMstIntf(CmBusMstIntf *busif){
     BusMstIntf=busif;
   }
   void OneStep(void){
     ...                                                         [2]
     BusMstIntf->AppendReadData(…);
     BusMstIntf->StartRead(…);                                   [3]
     ...
   }
}
```

FIG. 31B

Schematic Diagram

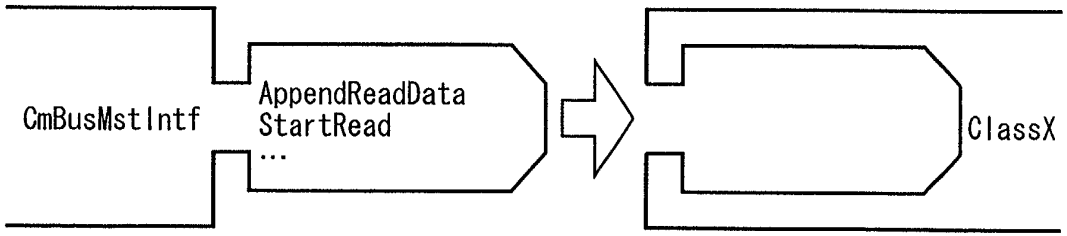

FIG. 32A

Example 1 of Statement of System

```
main(){
    ClassX ObjectX;                          [1]
    CmPciBusMstIntf PciBusMstIntf;           [2]
    ...
    ObjectX.ConnectBusMstIntf(&PciBusMstIntf);   [3]
    ...
}
```

FIG. 32B

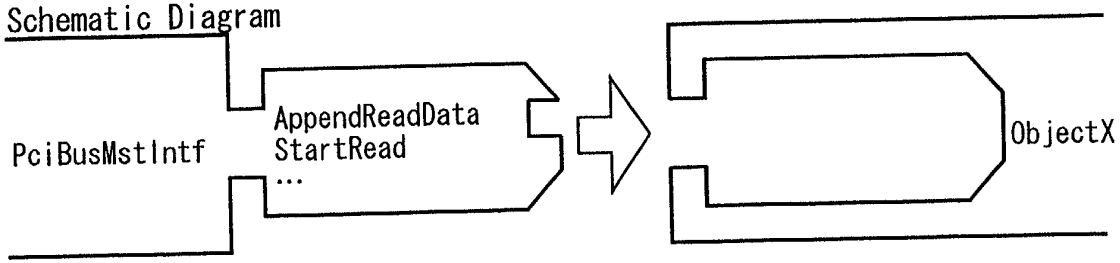

FIG. 33A

Example 2 of Statement of System

```
main(){
    ClassX ObjectX;
    CUserBusMstIntf UserBusMstIntf;
    ...
    ObjectX.ConnectBusMstIntf(User&BusMstIntf);
    ...
}
```

FIG. 33B

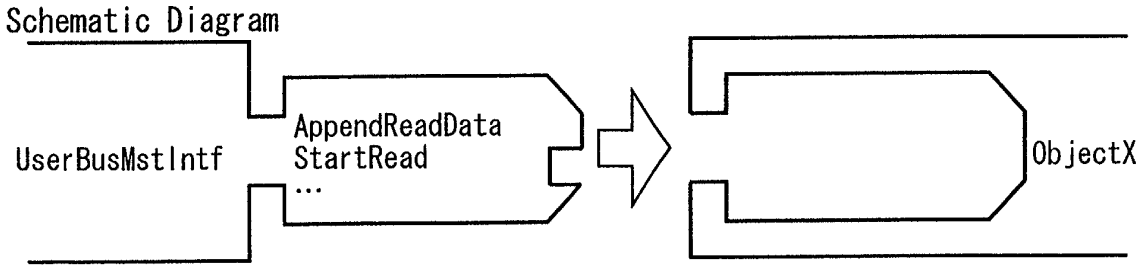

FIG. 34A

Statement of ClassY

```
class ClassY{                                          [1]
  CmComponent *PointerComponent
  void ConnectComponent(CmComponent *c){
    PointerComponent=c;
  }
  void OneStep(void){
    ...
    ...=PointerComponent->GetPinValue();               [2]
    ...
  }
};
```

FIG. 34B

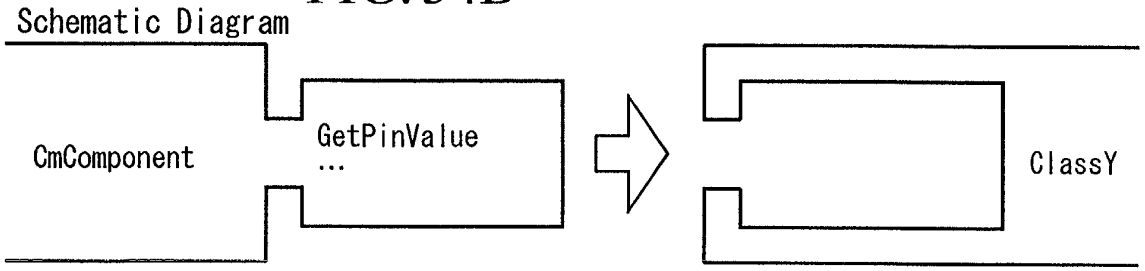

FIG. 35A

Example of Statement of System

```
main(){
  ClassY ObjectY;
  CUserCpu UserCpu; ——[1]
  ...
  Object.YConnectComponent(&UserCpu);
  ...
}
```

FIG. 35B

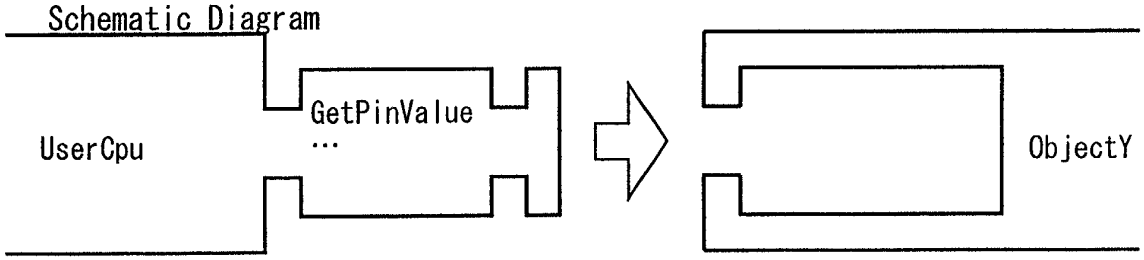

FIG. 36A

Statement in C++

```
main(){
  CmBusSlave BusSlave;                                    [1]
  CmBusMstIntf BusMstIntf;
  ...
  BusSlave.ConnectBusSlvIntf(&BusMstIntf);  //Error
  ...
}
```

Schematic Diagram FIG. 36B

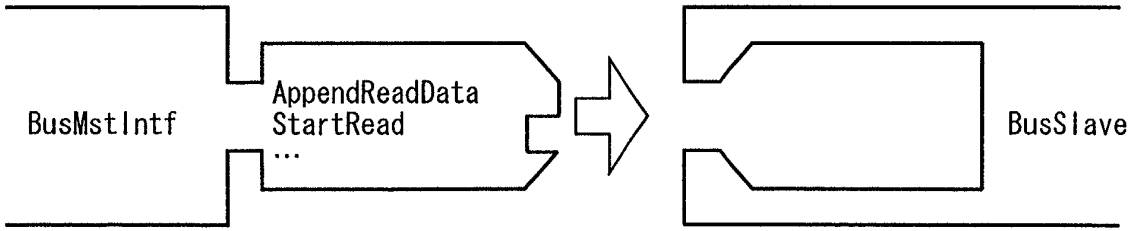

FIG. 37A

Statement in HDL

```
module C;
    wire c1,…
    ModuleA  A(.a1(c1),…);
    ModuleB  B(.b1(c1),…);
endmodule
```

FIG. 37B